Elmar Schömer
Ann-Christin Wörl

7. Übungsblatt

Abgabe: Dienstag, der 12.12.2023, 14:00 Uhr

Aufgabe 1: Rekursion I (5+5+10+5+10 Punkte)

1. Schreiben Sie eine rekursive Funktion `sum(n,A)` zur Berechnung der Summe der Elemente eines Feldes $A = [a_0, a_1, \ldots, a_{n-1}, \ldots]$ mit $n \leq len(A)$.

$$\sum_{i=0}^{n-1} a_i = sum(n, A) = \begin{cases} sum(n-1, A) + a_{n-1} & \text{für } n > 0 \\ 0 & \text{für } n = 0 \end{cases}$$

2. Schreiben Sie eine rekursive Funktion `ggT(a,b)` zur Berechnung des größten gemeinsamen Teilers von zwei natürlichen Zahlen $a, b \in \mathbb{N}$.

$$ggT(a, b) = \begin{cases} ggT(b, a\%b) & \text{für } a \geq b > 0 \\ a & \text{für } b = 0 \end{cases}$$

3. Begründen Sie, warum die folgende Python-Funktion `f(n)` bei Eingabe einer natürlichen Zahl, folgende mathematische Funktion $f : \mathbb{N} \to \mathbb{N}$ (auf umständliche Weise) berechnet:

$$f(n) = \begin{cases} 91 & \text{für } 0 \leq n \leq 100 \\ n - 10 & \text{für } n > 100 \end{cases}$$

```
1  def f(n):
2      if n > 100:
3          return(n-10)
4      return(f(f(n+11)))
5
6  print(f(42))
```

4. Die Collatz-Funktion $f : \mathbb{N} \to \mathbb{N}$ (siehe https://de.wikipedia.org/wiki/Collatz-Problem) kann man wie folgt rekursiv definieren:

$$f(n) = \begin{cases} 1 & \text{für } n = 1 \\ f(n/2) & \text{für } n \text{ gerade} \\ f(3n+1) & \text{für } n \text{ ungerade und } n > 1 \end{cases}$$

   Zeigen Sie experimentell, dass $\forall\ 1 \leq n \leq 10^6 : f(n) = 1$, indem Sie sowohl eine iterativ als auch eine rekursiv arbeitende Funktion `f(n)` schreiben.

5. Die folgenden Wörter sind Beispiele für Palindrome (siehe https://de.wikipedia.org/wiki/Palindrom): *stets, Rentner, Reliefpfeiler, Regallager, neben, Ehe, Neffen, Radar, Uhu.* Wenn man ein Palindrom von links nach rechts liest, ergibt sich das selbe Wort wie beim Lesen von rechts nach links. Schreiben Sie eine rekursive Funktion zur Überprüfung, ob eine gegebene Zeichenkette ein Palindrom ist.

Aufgabe 2: Rekursion II (10+5+5+5+10* Punkte)

Unter der Potenzmenge $\mathcal{P}(\mathcal{A})$ einer Menge A versteht man die Menge $\{X|X \subseteq A\}$ aller Teilmengen von A. Z.B.

$$\begin{array}{lcl} A & = & \{a,b,c\} \\ \mathcal{P}(\mathcal{A}) & = & \{\{\},\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\} \end{array}$$

1. Erläutern Sie die grundlegende Idee, wie es der nachfolgenden rekursiven Funktion `potenzmenge(n,A)` gelingt, die Potenzmenge einer n-elementigen Menge A zu berechnen.
2. Erklären Sie, wie das Ende der Rekursion erreicht wird und was in diesem Fall der korrekte Rückgabewert ist.
3. Erklären Sie im Hinblick auf die Speicherverwaltung, was in Zeile 7 passiert.
4. Verwenden Sie in der Funktion eine globale Variable zum Zählen der Elemente der Potenzmenge. Verifizieren Sie, dass die Mächtigkeit $|\mathcal{P}(A)| = 2^n$ ist, wenn die Menge A die Mächtigkeit $|A| = n$ besitzt.

```
1  def potenzmenge(n,A):
2      if n == 0:
3          return([[]])
4      P = []
5      for M in potenzmenge(n-1,A):
6          P.append(M)
7          P.append(M+[A[n-1]])
8      return(P)
9
10 print(potenzmenge(3,['a','b','c']))
```

Zusatzaufgaben:

1. Schreiben Sie eine Funktion `teilmengen(n,k,A)`, die für eine n-elementige Menge A alle k-elementigen Teilmengen berechnet.
2. Für die Zahl der k-elementigen Teilmengen einer n-elementigen Menge gilt folgende Rekursionsgleichung:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{für } n > 0 \text{ und } 0 < k \leq n, \quad \text{wobei } \binom{n}{0} = \binom{n}{n} = 1 \quad \text{für } n \geq 0$$

   Schreiben Sie eine Funktion `binomial(n,k)` zur rekursiven Berechnung der Binomialkoeffizienten.
3. Wie hoch ist die Laufzeit Ihrer Funktion bei der Berechnung von `binomial(100,50)`? Verwenden Sie die Memoisationstechnik, um die Laufzeit deutlich zu verbessern.

Aufgabe 3: Rundreiseproblem (5+15+5+10* Punkte)

Wir wollen das berühmte Rundreiseproblem (siehe https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden) auf naive Art und Weise lösen. Gesucht ist die kürzeste Rundreise durch n Städte in Rheinland-Pfalz. Die Entfernungen (Luftlinie) zwischen je zwei Städten können aus einer Tabelle entnommen werden. Eine einfache Lösung des Problems besteht darin, alle $(n-1)!$ vielen Rundreisen aufzuzählen, jeweils die Länge der Route zu bestimmen und sich die kürzeste Route zu merken.
Als Hilfe können Sie auf eine Funktion `allePermutationen` zurückgreifen, die in der Lage ist, alle Permutationen der Elemente eines gegebenen Feldes zu erzeugen.

1. Erklären Sie (nur) die grundlegende Idee hinter der Arbeitsweise der Funktion `allePermutationen`.
2. Schreiben Sie ein Programm zur Bestimmung der kürzesten Rundreise durch die sieben unten genannten Städte.
3. Welche Laufzeit Ihres Programmes würden Sie erwarten, wenn Sie die kürzeste Rundreise für 100 Städte berechnen wollten?

Zusatzaufgabe: Beweisen Sie die Korrektheit der Funktion `allePermutationen` zur Erzeugung aller Permutationen.

```
1  # Das eindimensionale Feld der Staedtenamen.
2  stadt = ['MZ', 'KO', 'TR', 'SP', 'WO', 'KL', 'LU']
3
4  # Das zweidimensionale Feld der Abstandstabelle.
5  dist = [[0, 61, 119, 77, 41, 71, 61],
6          [61, 0, 95, 129, 97, 101, 116],
7          [119, 95, 0, 137, 124, 87, 133],
8          [77,129, 137, 0, 35, 50, 17],
9          [41, 97, 124, 35, 0, 48, 19],
10         [71, 101, 87, 50, 48, 0, 49],
11         [61, 116, 133, 17, 19, 49, 0]]
12
13 def allePermutationen(n,A,P):
14     def swap(A,i,j):
15         tmp = A[i]
16         A[i] = A[j]
17         A[j] = tmp
18
19     if n == 0:
20         P.append([A[i] for i in range(len(A))])
21     for i in range(n-1,-1,-1):
22         swap(A,i,n-1)
23         allePermutationen(n-1,A,P)
24         swap(A,i,n-1)
25
26 P = []
27 allePermutationen(3,[1,2,3],P)
28 print(P)
```

Aufgabe 4: Rucksackproblem (15 Punkte)

Wir wollen das Rucksackproblem (siehe https://de.wikipedia.org/wiki/Rucksackproblem) auf naive Art und Weise lösen. Gegeben sind n Gegenstände. Der i-te Gegenstand besitzt den Wert w_i und das Gewicht g_i für $0 \leq i < n$. Die Tragelast des Rucksackes ist beschränkt durch das Gewicht G_{max}. Das Ziel ist es, den Rucksack mit einer möglichst wertvollen Auswahl von Gegenständen zu beladen, sodass das zulässige Gesamtgewicht nicht überschritten wird. Etwas formaler ausgedrückt ist eine Teilmenge $I \subseteq \{0, 1, \ldots, n-1\}$ gesucht, so dass

$$\sum_{i \in I} g_i \leq G_{max} \quad \text{und} \quad \sum_{i \in I} w_i \quad \text{maximal.}$$

Ein naiver Ansatz zur Lösung des Rucksackproblemes besteht darin, einfach alle Teilmengen der n-elementigen Menge der Gegenstände aufzuzählen und eine Teilmenge zu bestimmen, deren Gesamtgewicht zulässig ist und den größt möglichen Wert erreicht.
Implementieren Sie diese Vorgehensweise. Dabei können Sie die Funktion `potenzmenge` aus Aufgabe 2 zu Hilfe nehmen.

```
1  import random
2
3  random.seed(4242)
4
5  n = 22
6  Gewicht = [random.randint(20,100) for i in range(n)]
7  Wert = [random.randint(20,100) for i in range(n)]
8
9  Gmax = 500
10
11 print('Gewichte:',Gewicht)
12 print('Werte:',Wert)
13 print('Gmax:', Gmax)
```